



Week9: Deep Learning

SETH HUANG

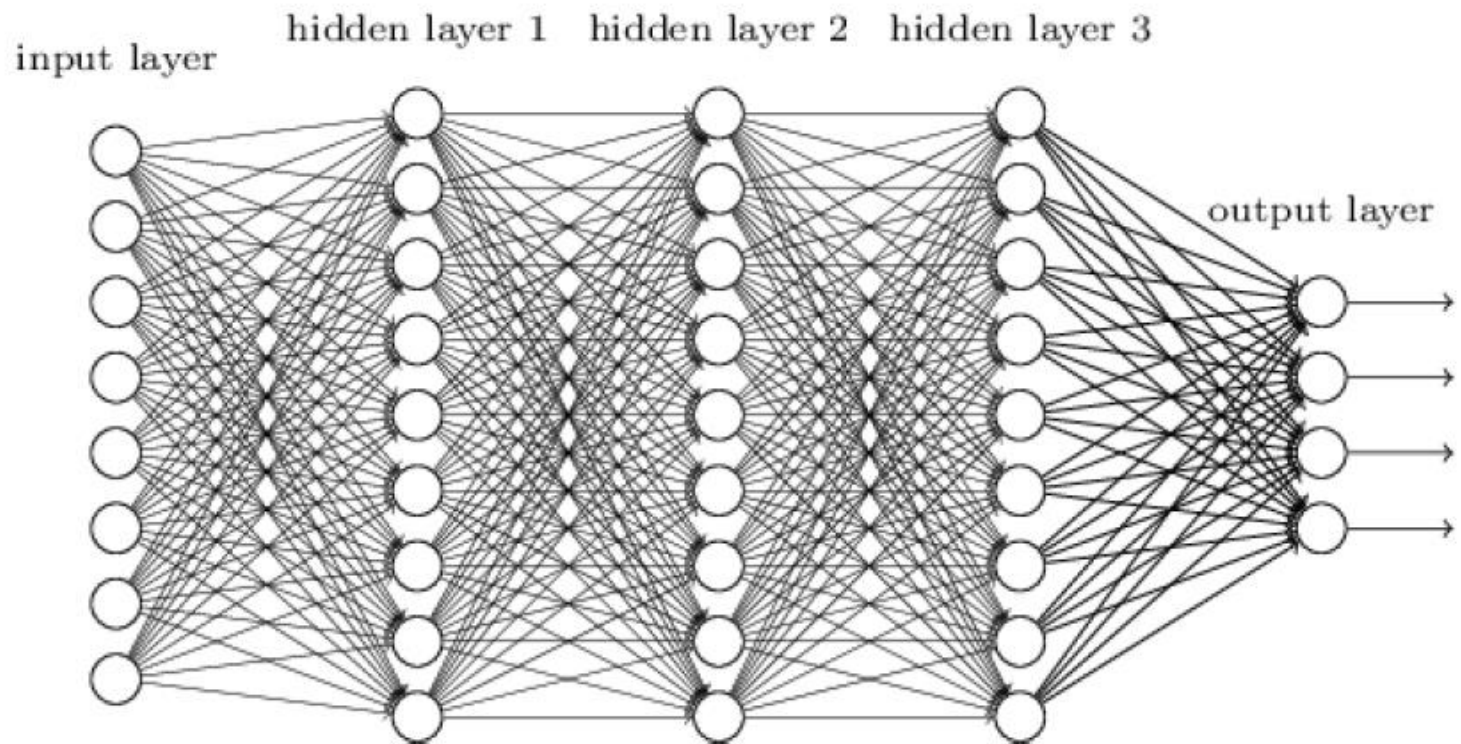
- DEEP NEURAL NETWORK

WE HAVE DISCUSSED THE FOLLOWING:

- Python programming language
- Neural network structure
- Weights, nodes, neurons
- Using the Python codes for MNIST (written digits) model training
- How to judge training data and test data
- The math behind training the model
- Regularization (L2) and why it helps
- Other regularization techniques (expanding data, L1, drop-out techniques)

=> We are getting very close to deep-learning

DEEP LEARNING WITH CONVOLUTIONAL NETWORKS



AS YOU SHOULD REMEMBER:

Each data is at 28X28 pixel image.

From before, we got about 98% accuracy on training for the MNIST dataset.

Note that, for a fully-connected network, input pixels that are far apart and close together are not important => treated equally

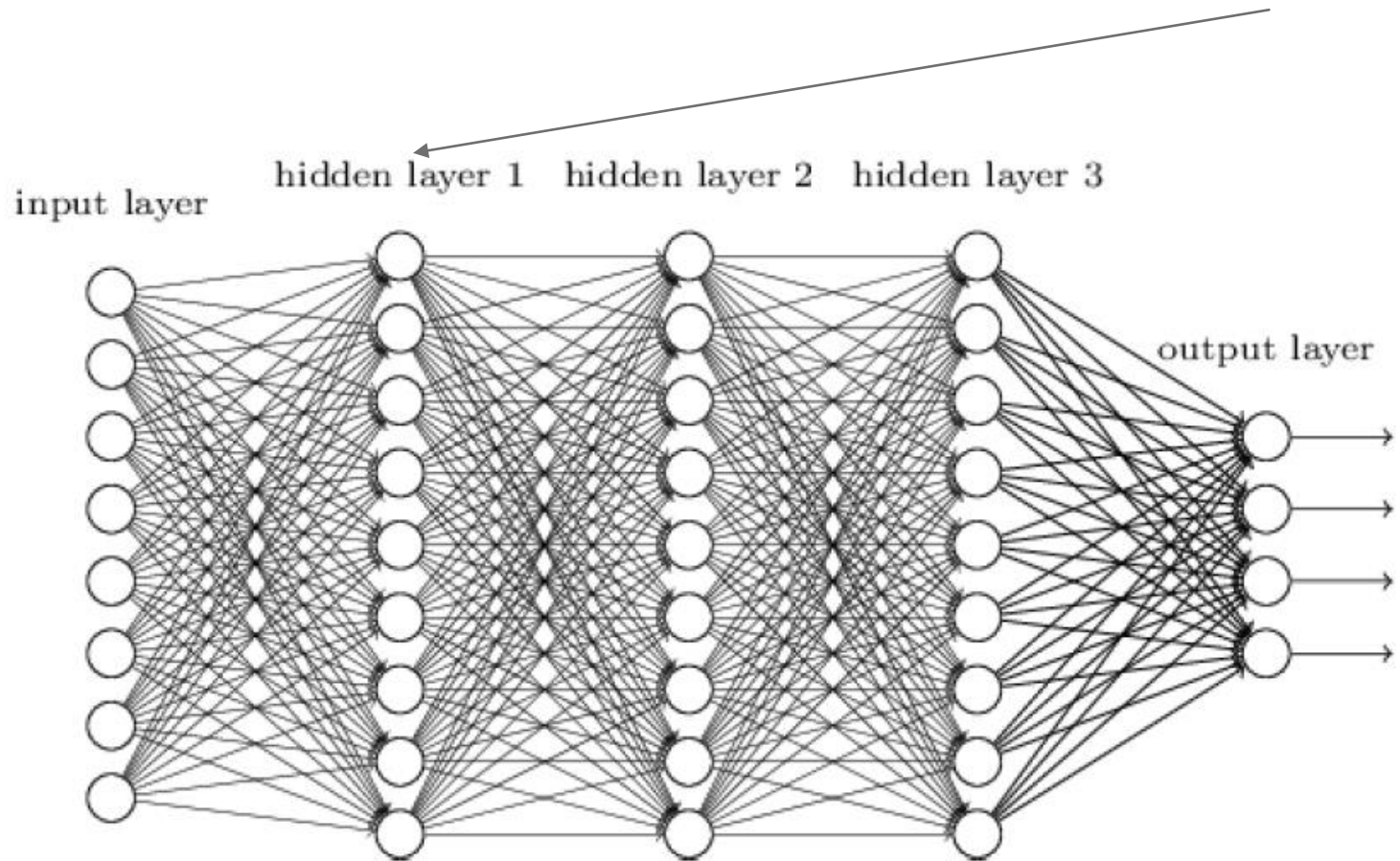
One special technique that takes advantage of that is called convolutional layer

THE COMPONENTS OF CONVOLUTIONAL NEURAL NETWORKS

- Local receptive fields
- Shared weights
- Pooling

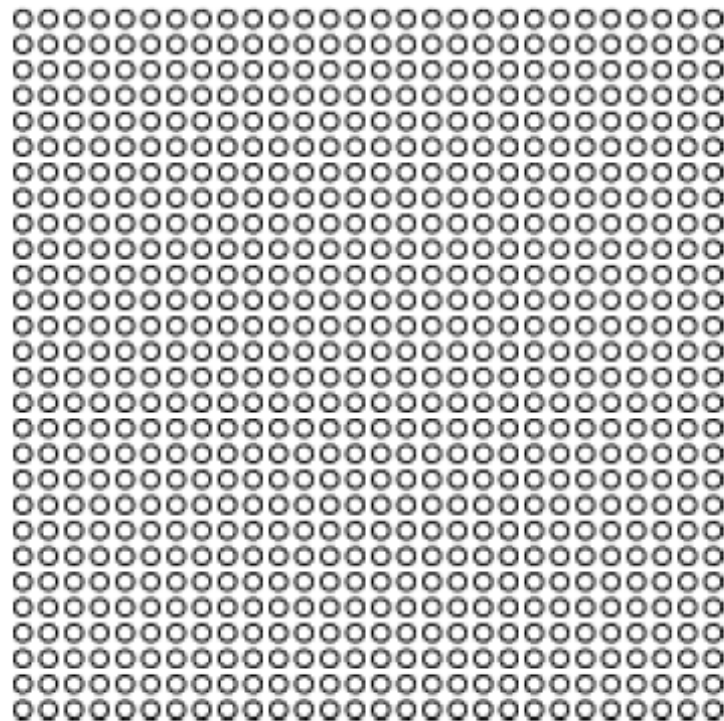
Keep in mind it's just another type of layer

CNN IS JUST PART OF THE STRUCTURE

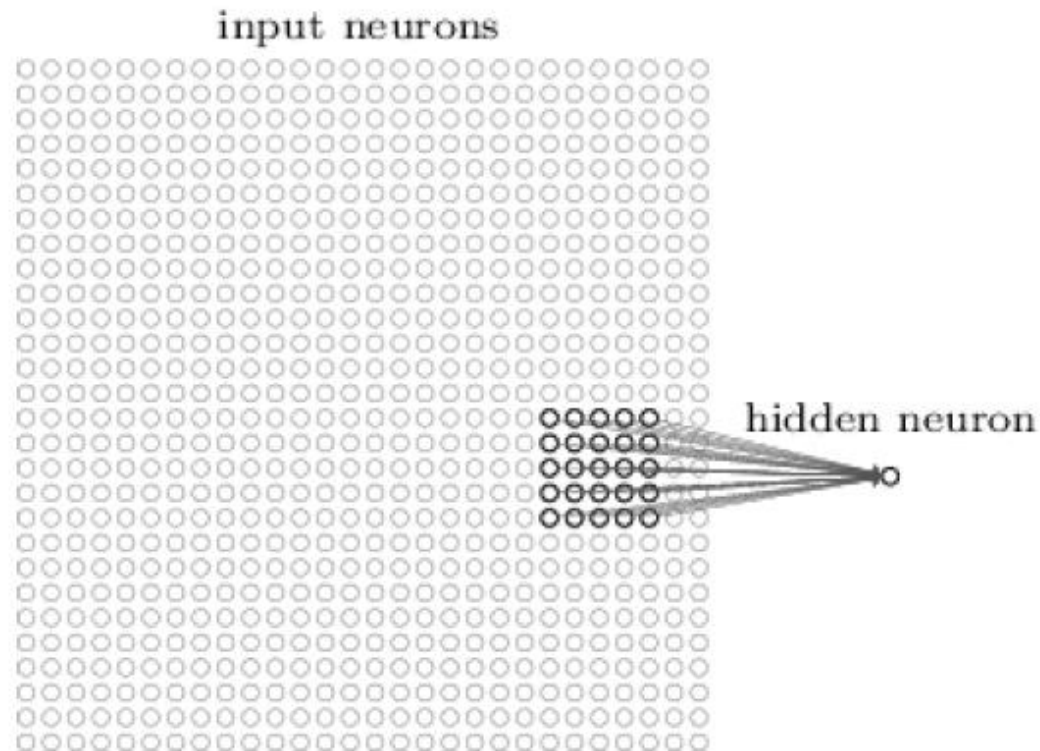


LET'S START SLOWLY: HERE ARE YOUR
INPUT NEURONS, 28X28

input neurons

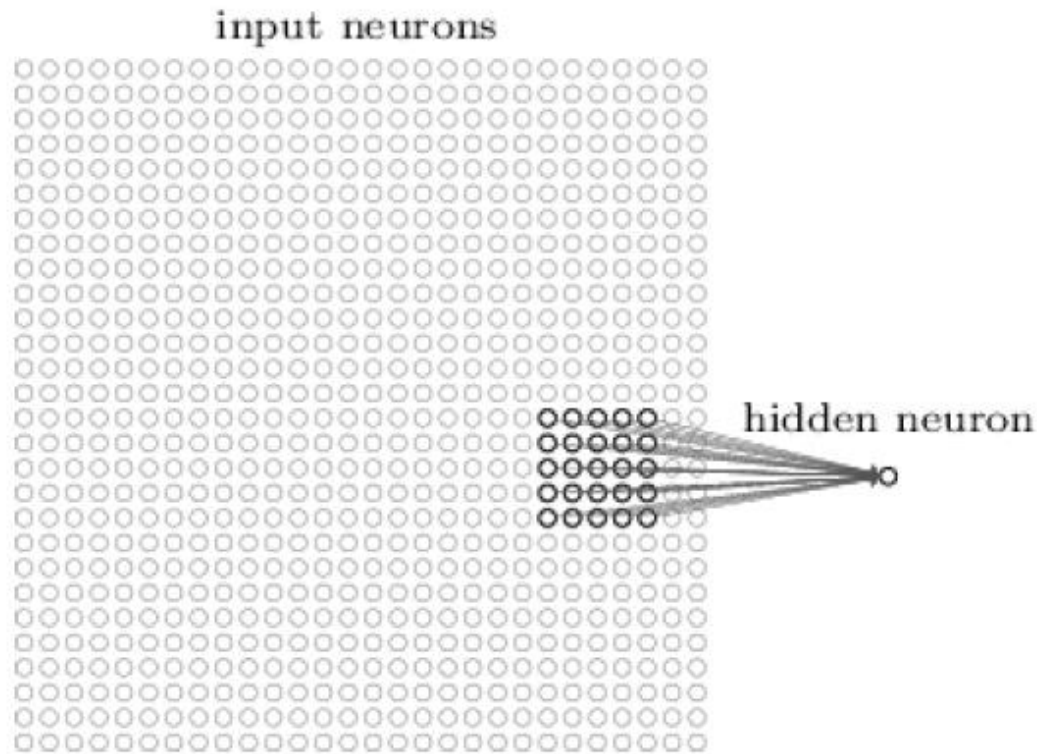


LOCAL RECEPTIVE FIELDS ARE SMALL RECTANGLES, SAY, 5X5, WHICH:



- Has 25 input pixels
- Will be connected to a small region of the input layer (784 pixels)

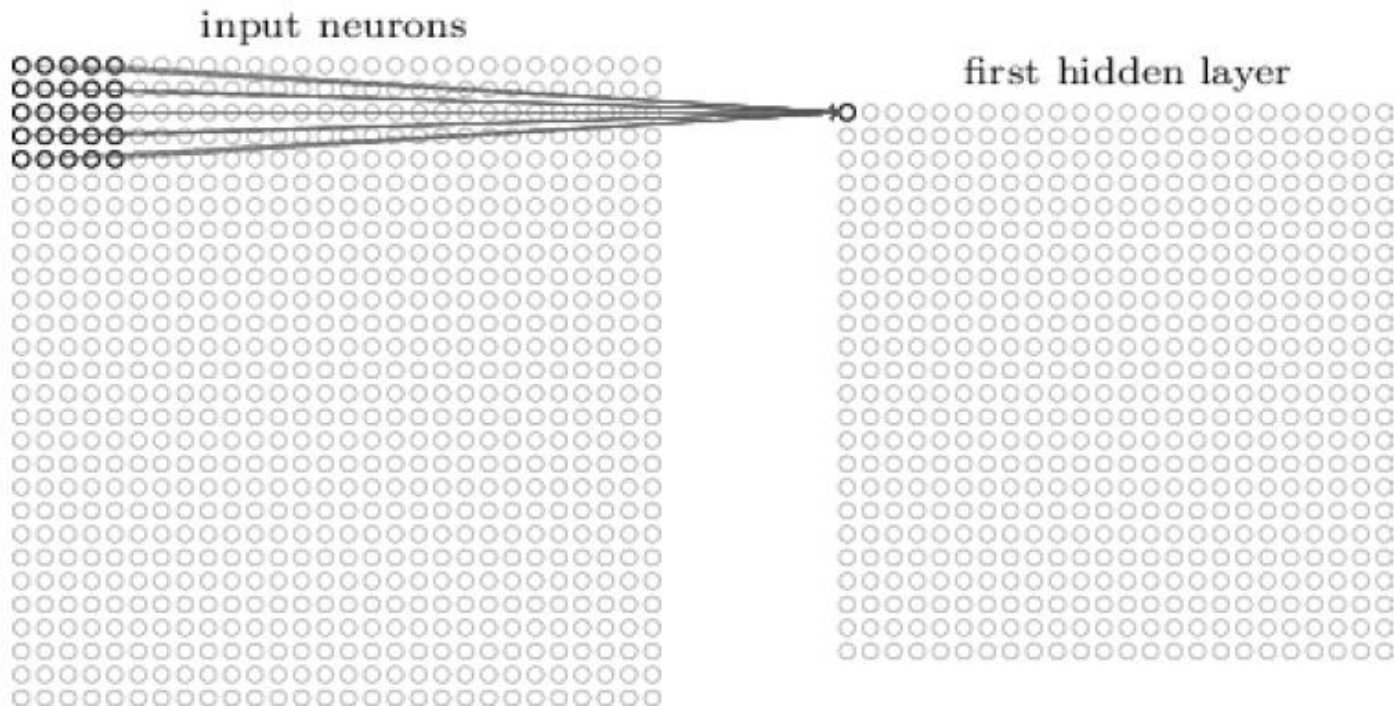
LET'S START SLOWLY: HERE ARE YOUR INPUT NEURONS, 28X28.



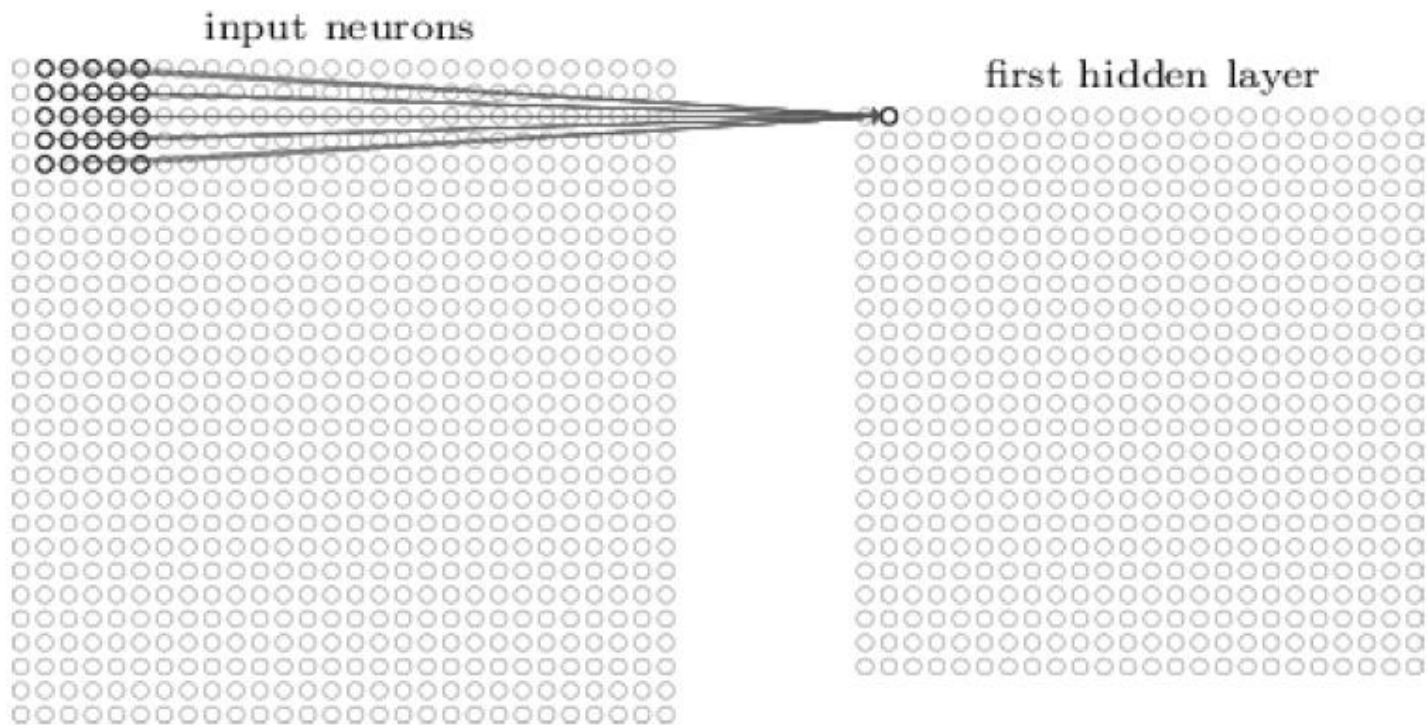
- Each connection has a weight
- This hidden neuron has a bias

This particular hidden neuron “analyzes” a particular part of the field

THEN IT SLIDES ACROSS THE ENTIRE
IMAGE

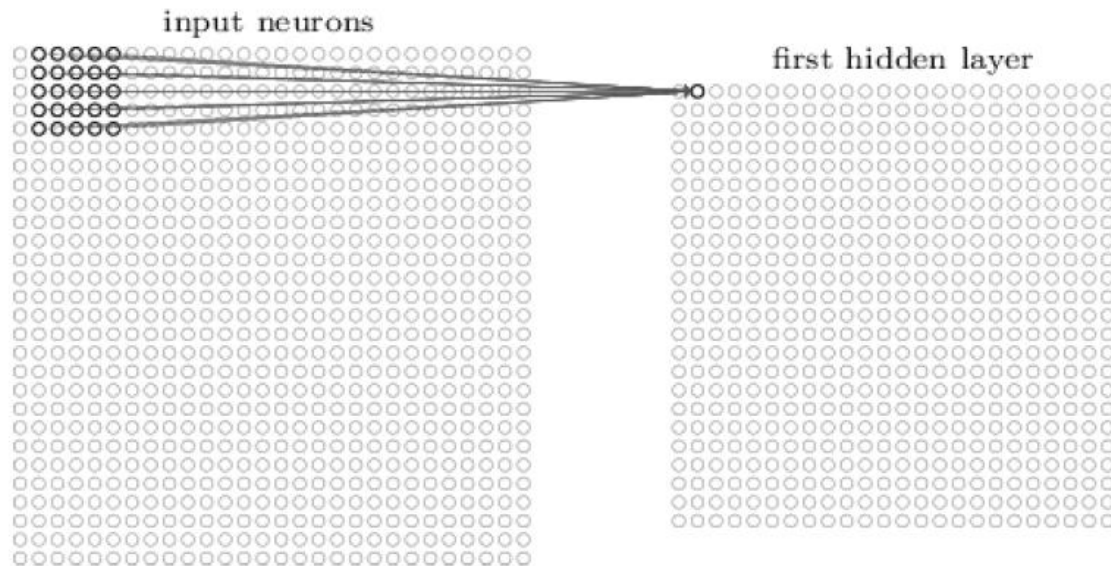


THEN IT SLIDES ACROSS THE ENTIRE
IMAGE



WE WILL HAVE 24 X 24 NEURONS IN THE HIDDEN LAYER

- You can choose a “stride length”
- You can choose how big the receptive field should be



SHARED WEIGHTS AND BIASES FOR EACH LOCAL RECEPTIVE FIELD

$$\sigma \left(b + \sum_{l=0}^4 \sum_{m=0}^4 w_{l,m} a_{j+l,k+m} \right) .$$

The sigma is an activation function like Sigmoid.
How many weights are there in the field?

=> Every part of the receptive field uses the same weights and bias!

⇒ This means every part of the receptive field detects the same features!

SPECIFICALLY:



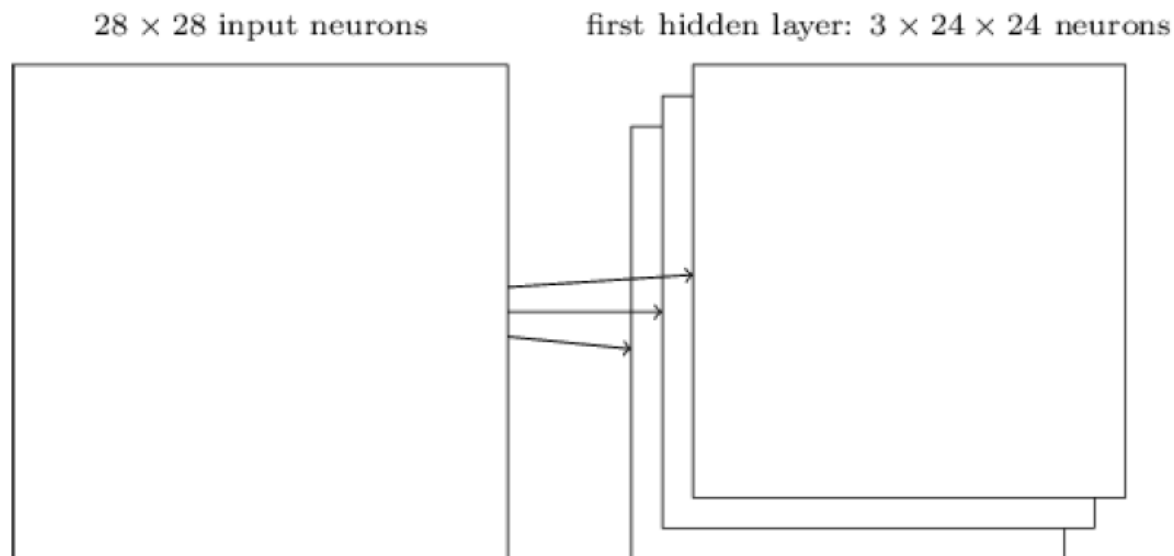
The first “layer” detects the same feature, such as a vertical edge.

It is good at “translating invariance” => Things are the same things even if they move a little bit

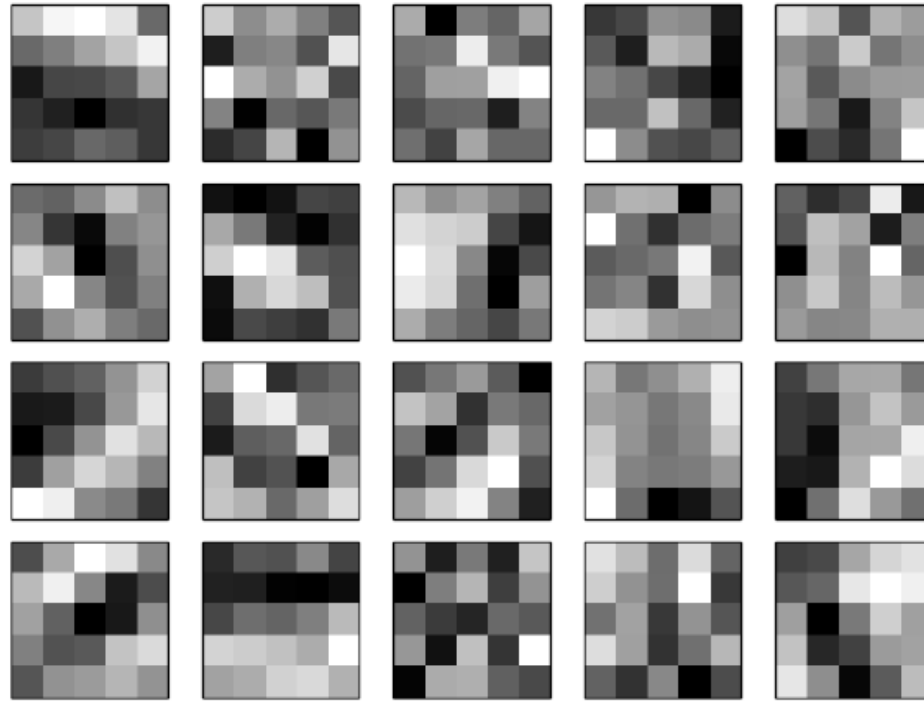
TO SUM UP:

The local receptive field from the input layer to the output layer is called a “feature map.”

For each feature map, the weights for the receptive fields are called “shared weights,” sometimes called a “kernel” or a “filter,” which simply remaps numbers into different numbers.

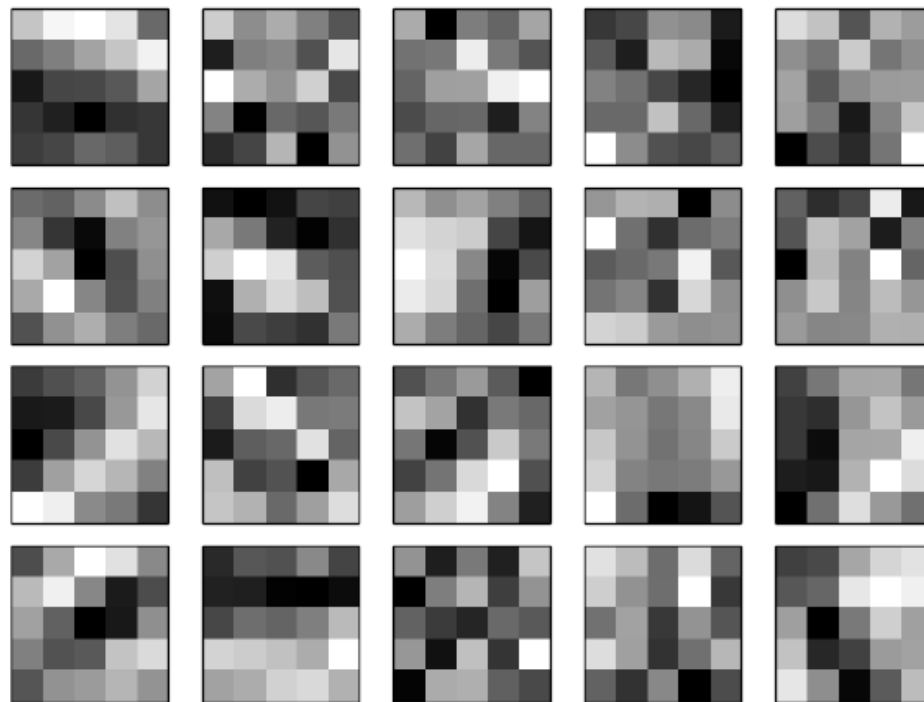


YOU CAN HAVE MORE THAN ONE FEATURE MAP – 20 FEATURE MAPS



Darker blocks mean higher weight = > it detects edges

JUDGING BY THEM, HOW IS IT RELATED TO THE SPECIAL FEATURES?



Features have regions that are darker or lighter

CONVOLUTIONAL LAYER:

~~- LOCAL RECEPTIVE FIELD~~

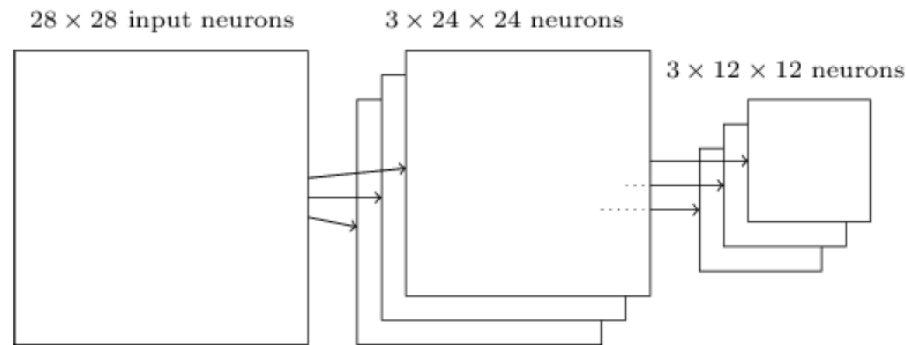
~~- SHARED WEIGHTS~~

- POOLING LAYER

POOLING LAYERS PUT ALL THE INFORMATION TOGETHER

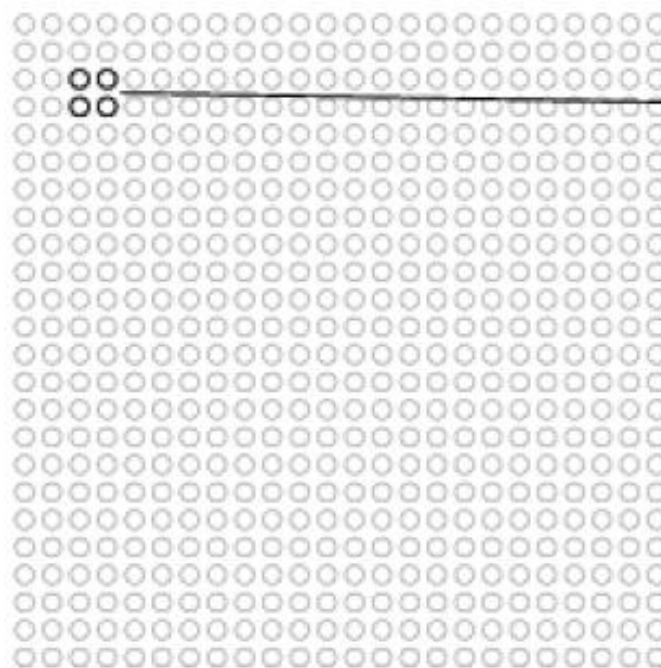
Pooling layers are always used right after the convolutional layer, so they can be treated as one package.

Pooling layers simplify the information in the output.

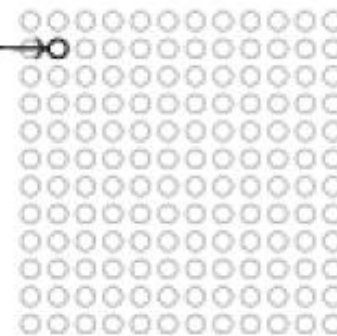


POOLING LAYER (ALMOST ALWAYS MAX POOLING) GETS INFO FROM THE HIDDEN NEURONS

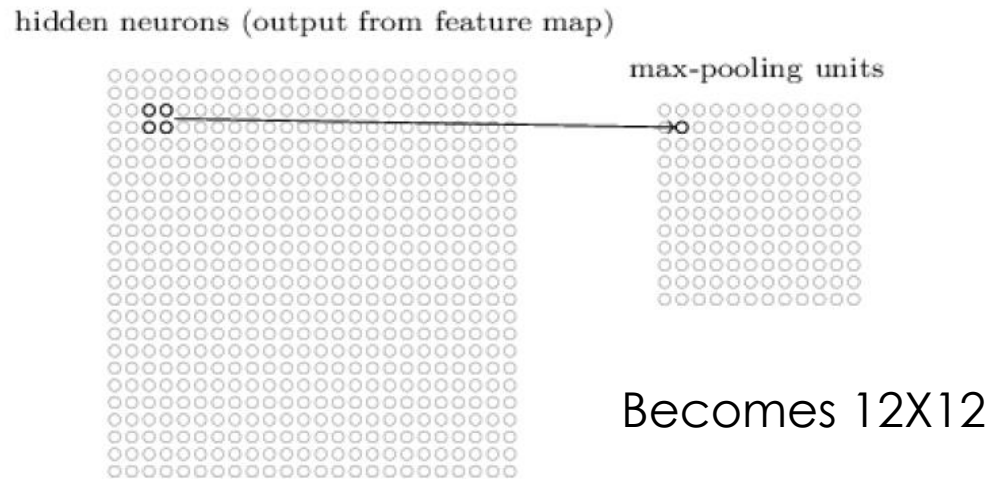
hidden neurons (output from feature map)



max-pooling units



POOLING LAYER (ALMOST ALWAYS MAX POOLING) GETS INFO FROM THE HIDDEN NEURONS



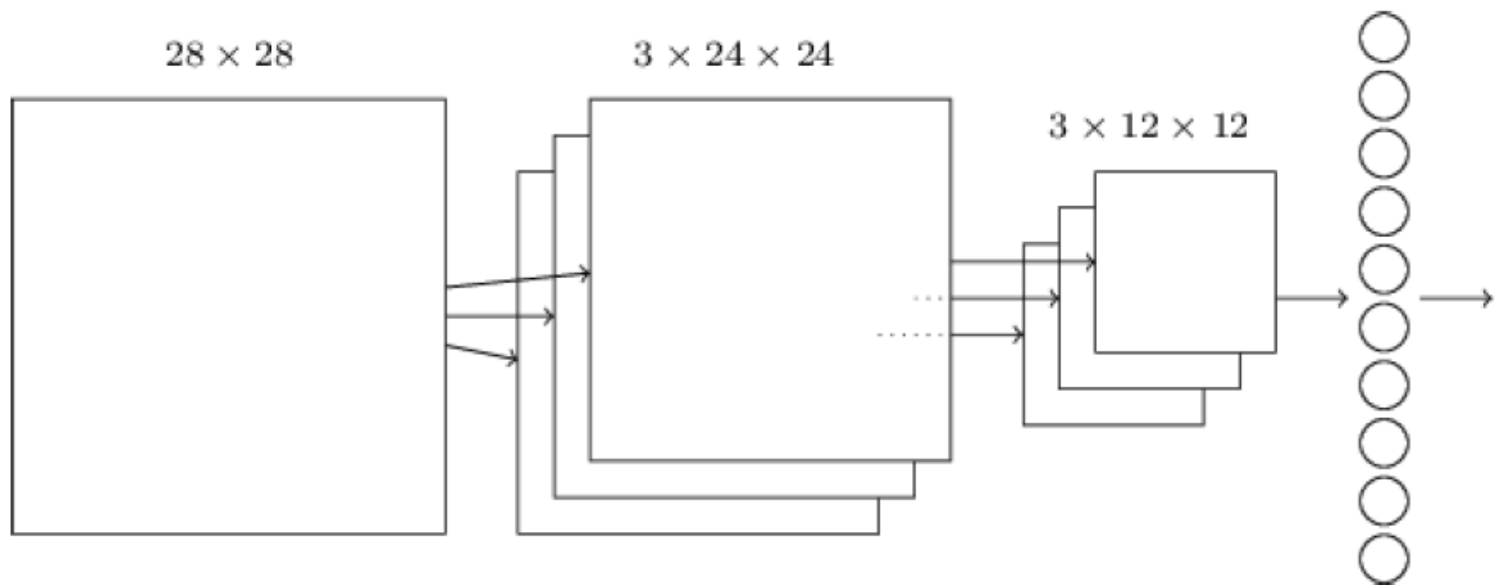
2x2 neurons
24X24 hidden layer

MAX-POOLING ASKS THE CONV LAYER: DID YOU SEE ANY SPECIFIC FEATURE?

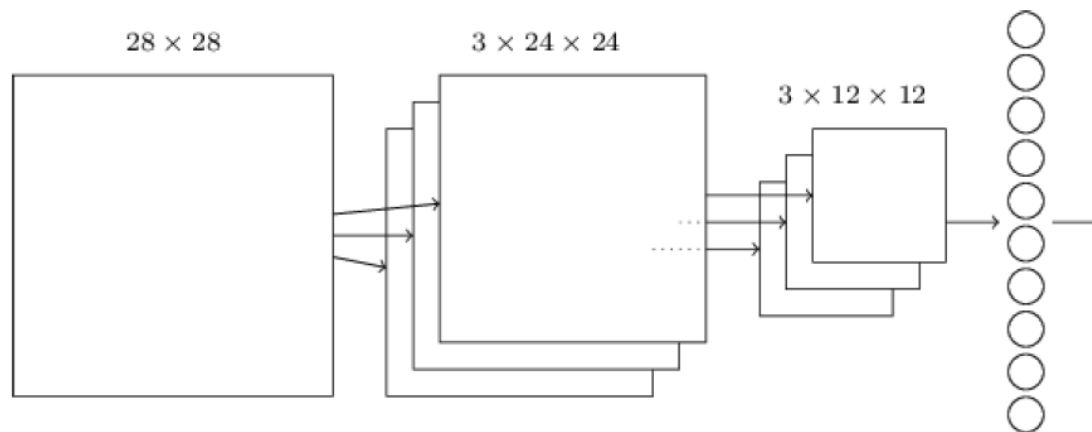
If the feature is found =>
the hidden neurons will generate numbers
⇒ Max-pooling summarizes (taking the largest number from each block)

Some people use Max, some people take the square root of the Sum (L2), and some people take the average. Most use Max.

PUTTING EVERYTHING TOGETHER

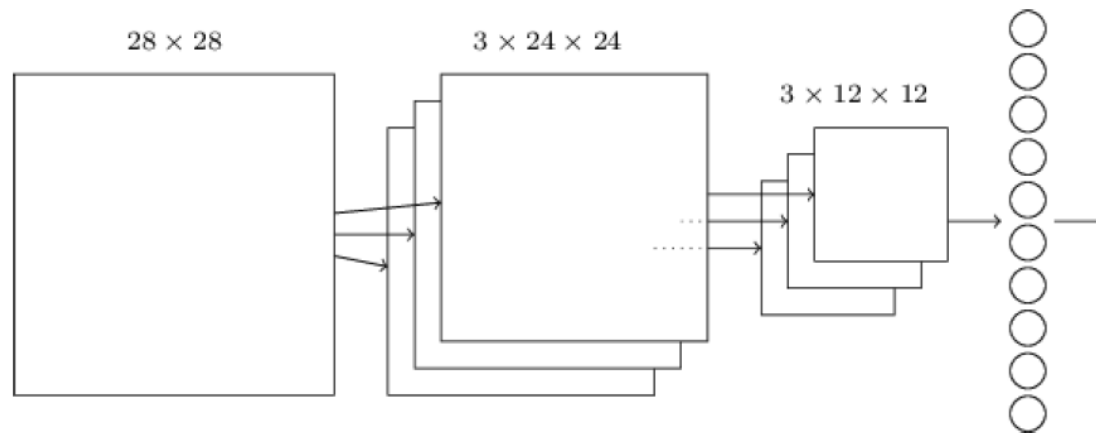


PUTTING EVERYTHING TOGETHER



- Begin with 28×28
- Followed by convolutional layer with 5×5 local receptive field (and a number of filters)
- You have $(28-5+1) = 24$ hidden “feature” neurons
- Max pooling layer with 2×2 regions result in $(24/2) = 12$ feature neurons for each filter
- In this case, there are 3 filters, so we have $3 \times 12 \times 12$ hidden feature neurons

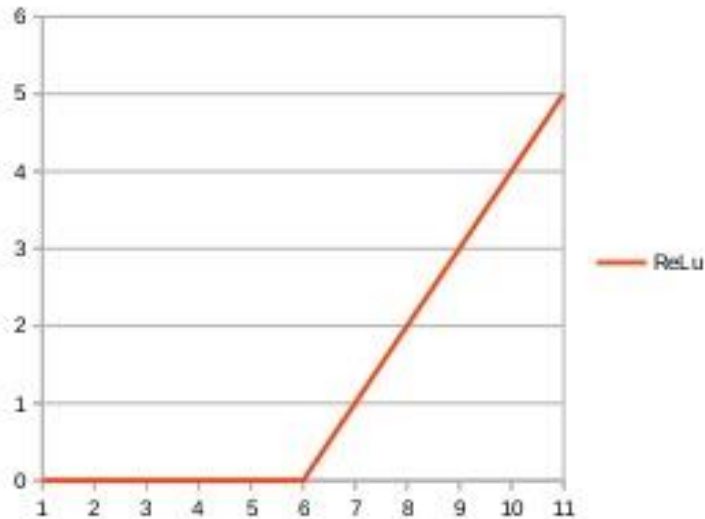
NOTE THAT THE LAST LAYER IS FULLY CONNECTED



From $3 \times 12 \times 12$ to the final layer, you bring all the information to one final “fully-connected” layer.

Activation Function Examples

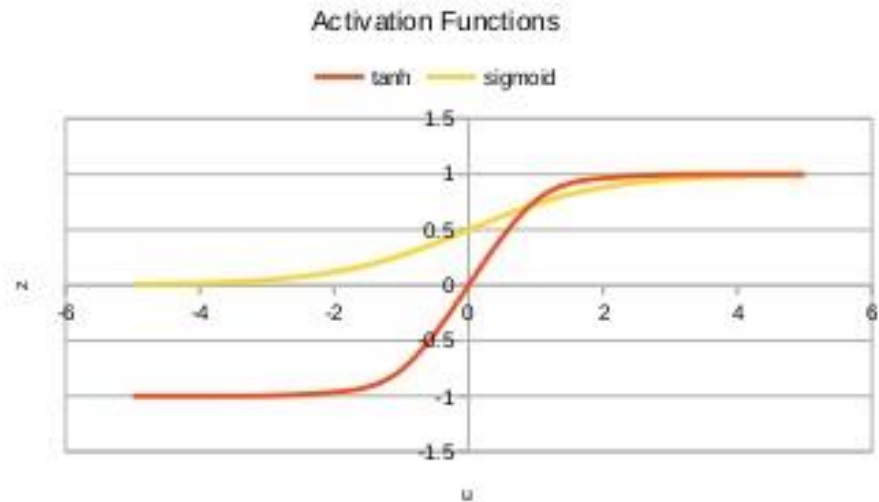
Relu



$$f(x) = \max(0, x)$$

tanh

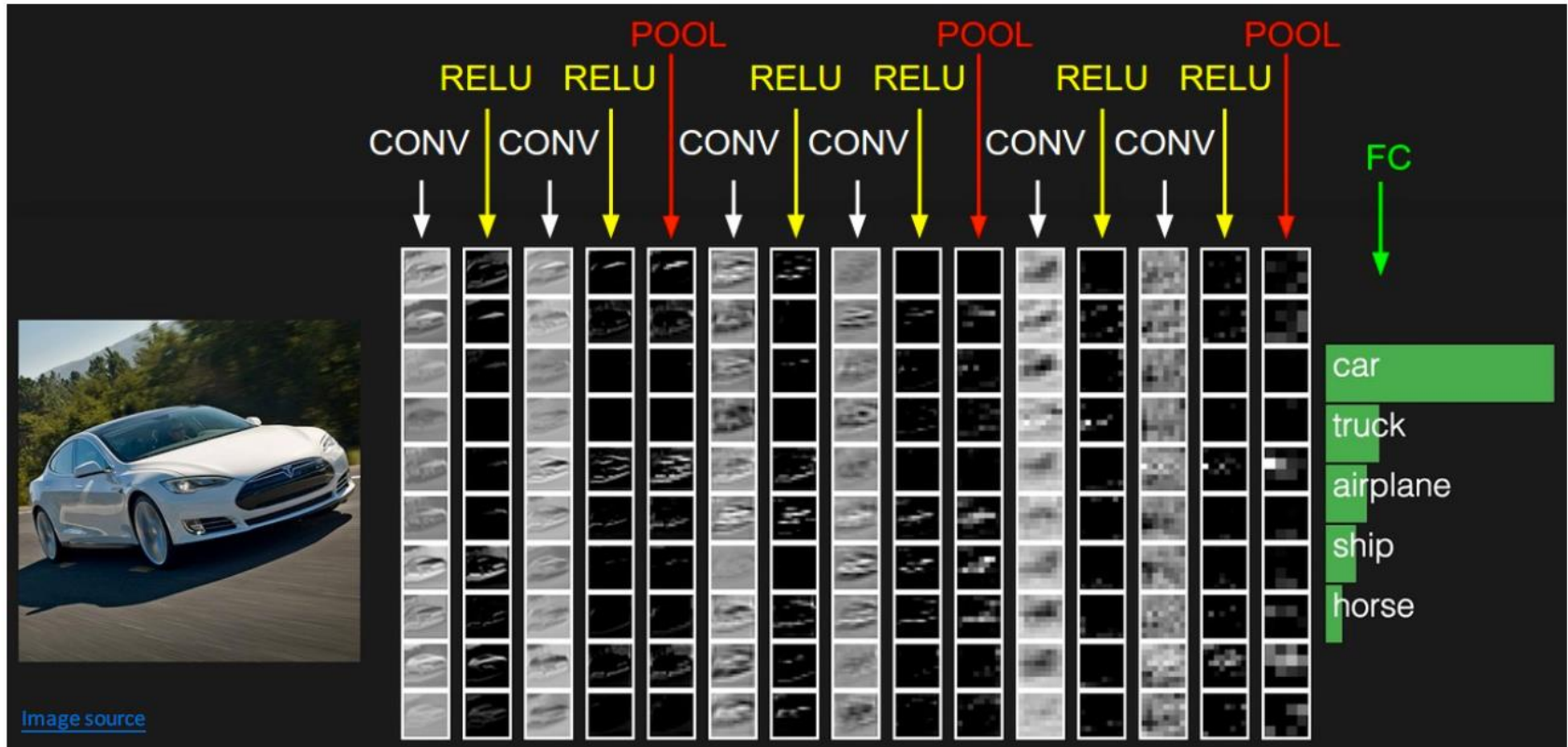
sigmoid



It's all you need to know in life!

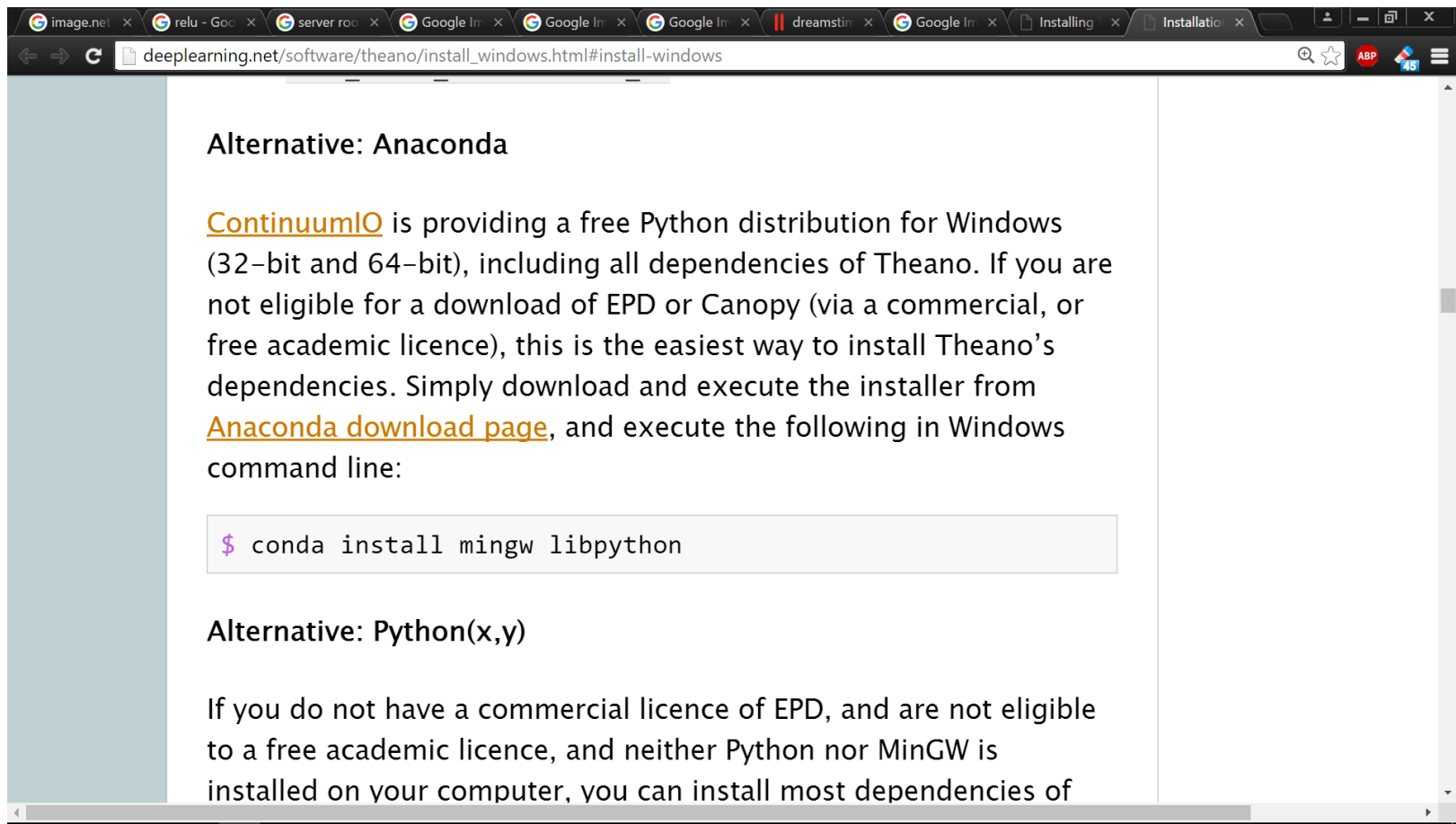
NOW YOU CAN UNDERSTAND THIS!

Convolutional Neural Network (CNN)



YOU SHOULD INSTALL THEANO (IT'S SOMETIMES NOT EASY)

http://deeplearning.net/software/theano/install_windows.html#install-windows



The screenshot shows a web browser window with multiple tabs. The active tab is titled 'deeplearning.net/software/theano/install_windows.html#install-windows'. The page content is as follows:

Alternative: Anaconda

[ContinuumIO](#) is providing a free Python distribution for Windows (32-bit and 64-bit), including all dependencies of Theano. If you are not eligible for a download of EPD or Canopy (via a commercial, or free academic licence), this is the easiest way to install Theano's dependencies. Simply download and execute the installer from [Anaconda download page](#), and execute the following in Windows command line:

```
$ conda install mingw libpython
```

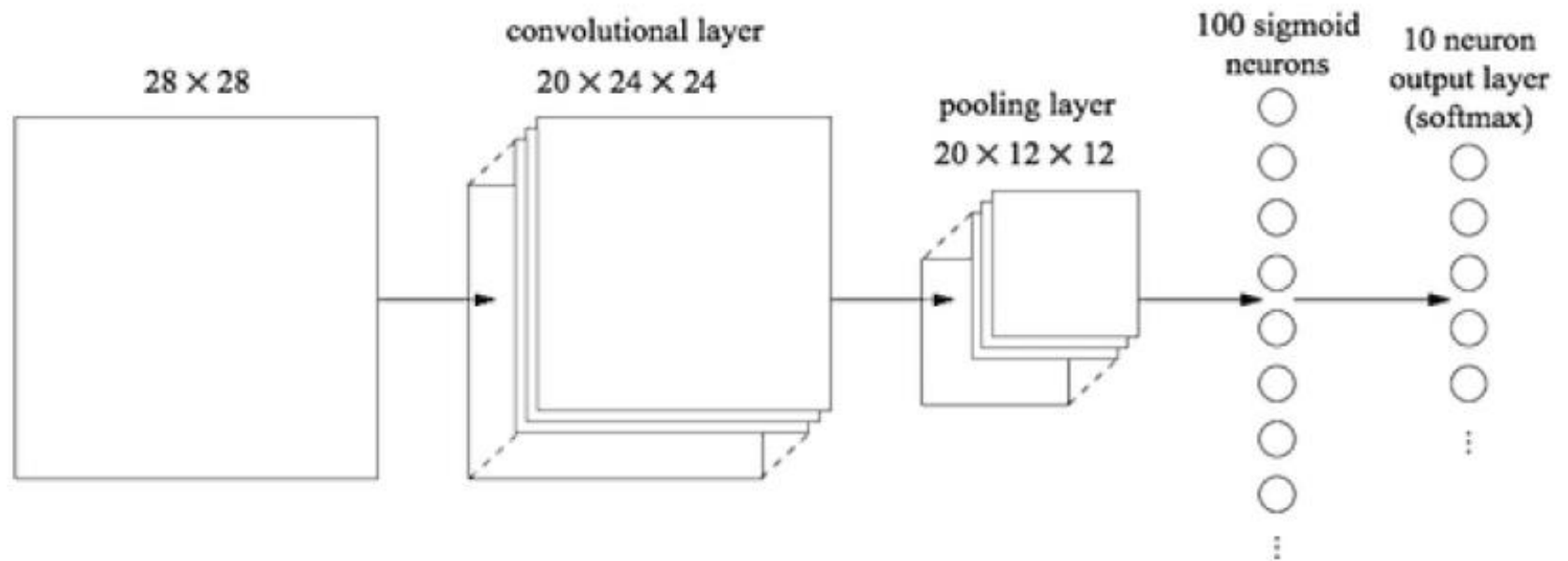
Alternative: Python(x,y)

If you do not have a commercial licence of EPD, and are not eligible to a free academic licence, and neither Python nor MinGW is installed on your computer, you can install most dependencies of

IN PRACTICE: THE BENCHMARK

```
>>> import network3
>>> from network3 import Network
>>> from network3 import ConvPoolLayer, FullyConnectedLayer, SoftmaxLayer
>>> training_data, validation_data, test_data = network3.load_data_shared()
>>> mini_batch_size = 10
>>> net = Network([
    FullyConnectedLayer(n_in=784, n_out=100),
    SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.1,
    validation_data, test_data)
```

IN PRACTICE: THE BENCHMARK



IN PRACTICE: OUR FIRST CNN!

```
>>> net = Network([
    ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                    filter_shape=(20, 1, 5, 5),
                    poolsize=(2, 2)),
    FullyConnectedLayer(n_in=20*12*12, n_out=100),
    SoftmaxLayer(n_in=100, n_out=10)], mini_batch_size)
>>> net.SGD(training_data, 60, mini_batch_size, 0.1,
             validation_data, test_data)
```